Performance Benefits of NVIDIA GPUs for LS-DYNA®

Mr. Stan Posey and Dr. Srinivas Kodiyalam

NVIDIA Corporation, Santa Clara, CA, USA

Summary:

This work examines the performance characteristics of LS-DYNA for the latest CPU and GPU technologies available. The results are provided for system configurations of workstations for finite element models that are relevant to current industry practice. The motivation for these studies was to quantify the parallel performance benefits of LS-DYNA for the latest generation GPU from NVIDIA, the Tesla 20-series (codenamed Fermi) for implicit finite element models with 100K and greater DOFs, and for static and dynamic response load conditions.

Keywords:

High Performance Computing, HPC, Tesla GPU, Computer Aided Engineering, CAE, Finite Element Analysis, FEA.

1. Introduction

Automotive, aerospace, defense and manufacturing industries continue to face growing challenges to reduce design cycle times and costs; satisfy global regulations on safety and environmental concerns; advance military programs; and respond to customers who demand high-quality, well-designed products. Because of these drivers, the desire for production deployment of LS-DYNA on high performance computing (HPC) systems for high-fidelity multi-physics simulations, design optimization, and other complex requirements, continue to push LS-DYNA workload demands of rapid single job turnaround and multi-job throughput capability for users with diverse application requirements. Additional HPC complexities arise for many LS-DYNA environments with the growth of multidiscipline CAE coupling of structural and CFD analyses, that all compete for the same HPC resources.

Current trends in HPC are moving towards the availability of several cores on the same chip of contemporary processors in order to achieve speed-up through the extraction of potential fine-grain parallelism of applications. The trend is led by GPUs, which have been developed exclusively for computational tasks as massively-parallel co-processors to the CPU. Today's GPUs can provide memory bandwidth and floating-point performance that are factors faster than the latest CPUs. Parallel efficiency and turnaround times continue to be important factors behind engineering decisions to develop LS-DYNA models at higher fidelity. A rapid CAE simulation capability from GPUs has the potential to transform current practices in engineering analysis and design optimization procedures.

2. LS-DYNA and HPC Considerations

Finite element analysis (FEA) software LS-DYNA from Livermore Software Technology Corporation (www.lstc.com) is a multi-purpose structural and fluid analysis software for high-transient, short duration structural dynamics, and other multiphysics applications. Considered one the most advanced nonlinear finite element programs available today, LS-DYNA has proved an invaluable simulation tool for industry and research organizations who develop products for automotive, aerospace, power-generation, consumer products, and defense applications, among others.

Sample LS-DYNA simulations in the automotive industry include vehicle crash and rollover, airbag deployment and occupant response. For the aerospace industry, LS-DYNA provides simulations of bird impact on airframes and engines and turbine rotor burst containment, among others. Additional complexities arise from simulations of these classes since they often require predictions of surface contact and penetration, models of loading and material behavior, and accurate failure assessment.

From a hardware and software algorithm perspective, there are roughly three types of LS-DYNA simulation characteristics to consider: implicit and explicit FEA for structural mechanics, and computational fluid dynamics (CFD) for fluid mechanics. Each discipline and associated algorithms have their inherent complexities with regards to efficiency and parallel performance, and also regarding modeling parameters.

The range of behaviors for the three disciplines that are addressed with LS-DYNA simulations, highlights the importance of a balanced HPC system architecture. For example, implicit FEA using direct solvers for static load conditions, requires a fast floating point performance and a high-bandwidth I/O subsystem for effective simulation turnaround times, and is in contrast to dynamic response, which requires very high rates of memory and I/O bandwidth with floating point speed as a secondary concern. In addition, FEA modeling parameters such as the size, the type of elements, and the load condition of interest all affect the execution behavior of implicit and explicit FEA applications.

Explicit FEA benefits from a combination of fast processors for the required element force calculations, and memory bandwidth for efficient contact resolution that is required for nearly every structural impact simulation. CFD also requires balance of memory bandwidth performance and fast floating point, but benefits most from parallel scalability. Each discipline has inherent complexities with regard to efficient parallel scaling, depending upon the particular parallel scheme of choice.

Implementations of both shared memory parallel (SMP) and distributed memory parallel (DMP) have been developed for LS-DYNA. The SMP version exhibits moderate parallel efficiency and can be used with SMP computer systems only, while the DMP version exhibits very good parallel efficiency across a range of shared and distributed memory systems. This DMP approach is based on domain

decomposition with a message passing interface (MPI) for communication between domain partitions, and is available for homogenous compute environments such as SMP systems or clusters.

Most parallel CAE software employ a similar DMP implementation based on domain decomposition with MPI. This method divides the solution domain into multiple partitions of roughly equal size in terms of required computational work. Each partition is solved on an independent processor core, with information transferred between partitions through explicit message passing in order to maintain the coherency of the global solution. LS-DYNA is carefully designed to avoid major sources of parallel inefficiencies, whereby communication overhead is minimized and proper load balance is achieved. In all cases the ability to scale I/O during the computation is critical to overall scalability in a simulation.

3. GPU-Parallel CAE

The continual increase in CPU speeds has limits due to power and thermal constraints with processors now having multiple cores. To achieve boosts in performance without increasing clock speeds parallelism must be developed. This parallelism can come in the form of task parallelism, data parallelism, or perhaps a combination of the two. Common methods for implementing parallelism are explicit message-passing using an MPI library for either distributed or shared memory systems, and OpenMP for shared memory systems. A hybrid method is also possible with OpenMP used on multi-core and multiprocessor nodes and MPI used among the nodes.

Although parallel applications that use multiple cores are a well established technology in computational science and engineering (CSE), a recent trend towards the use of Graphics Processing Units (GPUs) to accelerate CPU computations is emerging. In this heterogeneous computing model the GPU serves as a co-processor to the CPU. The need for high performance and the parallel nature of CSE problems has led GPU designers to create current designs with hundreds of cores. Today GPUs and software development tools are available for implementing more general applications that use the GPU not for graphics but for applications such as CAE and others where computations are needed to be completed as fast as possible.

Figure 1 shows the NVIDIA Fermi-based GPU architecture, which features up to 512 CUDA cores. A CUDA core executes a floating point or integer instruction per clock for a thread. The 512 cores are organized in 16 streaming multiprocessors (SMs) of 32 cores each. The GPU has six 64-bit memory partitions, for a 384-bit memory interface, supporting up to a total of 6 GB of GDDR5 DRAM memory. A host interface connects the GPU to the CPU via PCI-Express. The Fermi architecture is the first GPU to offer neccessary double precision performance for CAE applications, and can reach peak speeds of about 500 GFLOPS or ½ the speed of the single precision TFLOP performance.

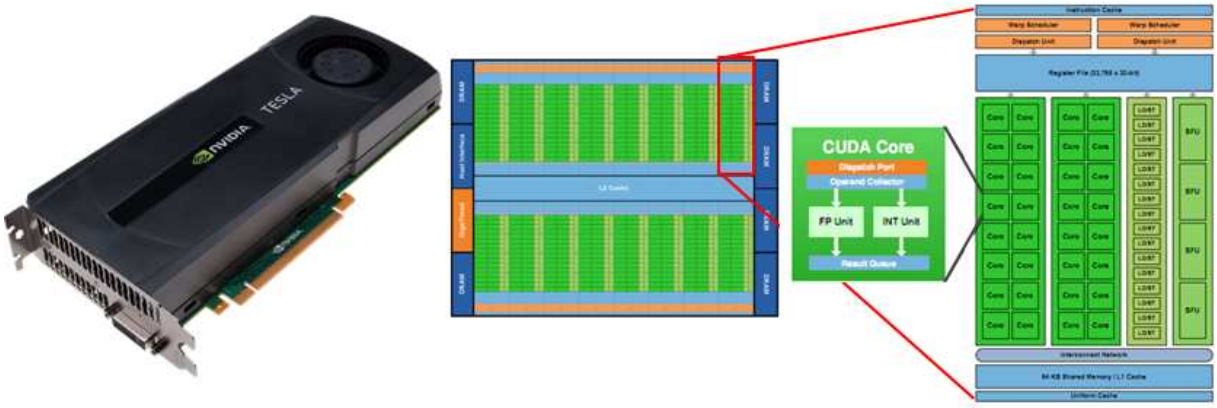

Figure 1: Image and Lay-out Description of the NVIDIA Tesla 20-Serieis Fermi GPU Architecture

Much work has recently been focused on GPUs as devices that can be used in general-purpose computing. A GPU can produce a very high FLOPS (floating-point operations per second) rate if an algorithm is well-suited for the device. There have been several studies illustrating the acceleration of scientific computing codes that is possible by using GPUs. Despite the tremendous performance gains possible with GPUs, relatively few commercial CAE software has yet to make use of them, and those

that have, demonstrate nominal overall gains of 2x over a multi-core CPU. This is mostly due to the current GPU focus on linear equation solvers rather than complete CAE code implementations.

The GPU was originally designed for graphics and the majority of this computation involves computing the individual color for each pixel on a screen. If we think of each pixel as being like a quadrilateral element, computing the pixel colors will be similar to the computations on a structured mesh. There is a very regular, orderly data access pattern with neighbors easily computed by simple offsets of indices. However, the majority of commercial CAE use some form of an unstructured mesh often with triangles in 2D or tetrahedral in 3D. These meshes lead to an irregular and somewhat disorderly data access pattern which is not particularly well suited to the memory system of a GPU.

Through a careful process of analyzing the relation between finite elements and vertices, and taking advantage of the large amount of available processor performance on a GPU, techniques can be applied that partitions and sorts the mesh in such a way that the data access pattern becomes much more regular and orderly. The preprocessing of the mesh connectivity is a one-time step performed just before the main computation begins and can require a negligible amount of compute time while significantly increasing the performance of the equation solver.

Shared memory is an important feature of the GPU and is used to avoid redundant global memory access among threads within a block. The GPU does not automatically make use of shared memory, and it is up to the software to explicitly specify how shared memory should be used. Thus, information must be made available to specify which global memory access can be shared by multiple threads within a block. For structured grid based solvers, this information is known up-front due to the fixed memory access pattern of such solvers, whereas the memory access pattern of unstructured grid based solvers is data-dependent.

Algorithm design for optimizing memory access is further complicated by the number of different memory spaces the developer must take into consideration. Unlike a CPU the memory accesses are under the full and manual control of the developer. There are several memory spaces on the GPU which in turn is connected to the CPU memory. Different memory spaces have different scope and access characteristics: some are read-only, some are optimized for particular access patterns. Significant gains (or losses) in performance are possible depending on the choice of memory usage.

Another issue to be considered for GPU implementation is that of data transfers across the PCI-Express bus which bridges the CPU and GPU memory spaces. The PCI-Express bus has a theoretical maximum bandwidth of 4 or 8 GB/s depending on whether it is of generation 1 or 2. When this number is compared to the bandwidth between the GPU's on-board GDDR3 memory and the GPU multi-processors (up to 141.7 GB/s), it becomes clear that an algorithm that requires a large amount of continuous data transfer between the CPU and GPU will unlikely achieve good performance.

For a FEA solver, the obvious solution is to limit the size of the domain that can be calculated so that all of the necessary data can be stored in the GPU's main memory. Using this approach, it is only necessary to perform large transfers across the PCI-Express bus at the start of the computation (geometry) and at the end (final flow solution). High-end NVIDIA GPUs offer up to 6 GB of main memory by the end of 2010, sufficient to store all the data needed by most commercial parallel CAE software, so this restriction is not a significant limitation.

Over the past 10 years, CAE has become increasingly reliant on clusters of multiprocessors to enable more detailed simulations within design time frames. For this reason, the scalability of a solver across multiple processors can be equally important as its single-processor performance. A potential problem with increasing the single-processor performance by an order of magnitude is that the multi-processor performance suffers since the time required to exchange boundary information remains roughly constant. When operating in parallel across multiple GPUs, some boundary information must be transferred across the PCI-Express bus at the end of each time step. However, with implementation of a low surface-to-volume ratio in mesh partitioning, this data transfer need not be a bottle-neck.

The relative performance of processors vs. memory over the past few decades has extended to more than 3 orders of magnitude. CPUs have gone to great lengths to bridge the gap between processor and memory performance by introducing instruction and data caches, instruction level parallelism, and so forth. And although GPUs offer a different approach in terms of hiding memory latency because of

their specialization to inherently parallel problems, the fact remains that processor performance will continue to advance at a much greater rate than memory performance. If we extrapolate out, without any fundamental changes in memory, processors will become infinitely fast relative to memory, and performance optimization will become solely an exercise in optimizing data movement.

4. LS-DYNA GPU Performance Study

Performance and parallel efficiency for LS-DYNA is dependent upon many specifics of a system architecture and the geometry and conditions of the simulation. Structural FEA simulations in LS-DYNA often contain a mix of materials and finite elements that can exhibit substantial variations in computational expense, which may create load-balance complexities. The ability to efficiently scale to a large number of processors is highly sensitive to load balance quality of computations. This study of LS-DYNA performance will not yet consider multi-node parallel distributed characteristics, and instead focus on the parallel efficiency that GPUs can provide for the single node computations.

GPU development of LS-DYNA has been limited to implicit modelling at the time of this paper, and specifically the use of a multi-frontal sparse direct solver that usually requires out-of-memory solution processing. This occurs because the stiffness matrix that must be factorized is typically much larger than the allowable memory for a particular server or set of cluster nodes. The model for the study however is small enough to keep in-memory and is comprised of 230K DOFs. The geometry is a set of concentric cylinders and a description of the model and conditions is provided in Figure 2. The system used for the study contained an Intel Xeon 5500 Nehalem based CPU with dual sockets of 4 cores each for a total of 8 cores, and a Tesla C2050 GPU with 448 cores and 3 GB of main GPU memory.

For the CYLOP5E6 model the matrix is symmetric with a rank of 760320, and its diagonal and lower triangle contain 29213357 non-zero entries. After reordering with METIS, it required 7.104E+12 operations to factor the matrix, and the resulting factored matrix contains 1.28E+09 entries.

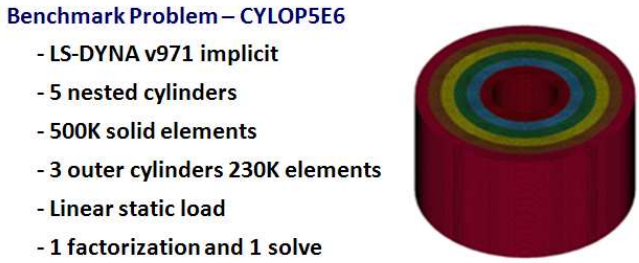

Figure 2: Description of the LS-DYNA implicit model CYLOP5E6 for tests of GPU acceleration

Figure 3 shows the total time required for LS-DYNA when factorizing the matrix, as a function of the number of cores employed, both with and without the GPU. The dual socket Nehalem host sustains 10.3 GFlop/s when using one core, and 59.7 GFlop/s when using all eight. When the GPU is employed, the benchmark performs 6.57E+12 operations or 92% of the total required, and sustained 98.1 GFlop/s peak. The overall performance with the GPU improves to 61.2 GFlop/s when one host core is used, and 79.8 GFlop/s with all eight.

The results provided in Figure 3 demonstrate that a GPU could substantially accelerate LS-DYNA when comparing wallclock times. In the case of 1 core operating with the GPU, the speedup is nearly 5x, but note that the optimal use of the combined CPU+GPU solution is at 4 cores of the CPU. At this level the acceleration is 2.4x and provides a total job time of just 240 seconds. This time can be lowered slightly to 215 seconds with the use of the 4 remaining cores, but is not the optimal use and could be applied to a 2nd GPU and 2nd job for improved CPU+GPU efficiency.

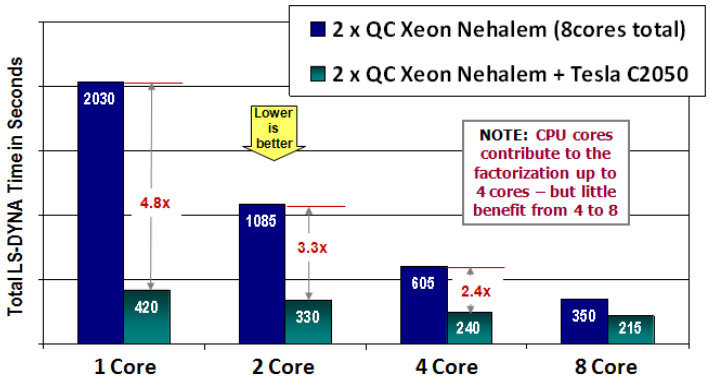

Figure 3: Performance results of end-to-end LS-DYNA wallclock time for model CYLOP5E6

5. Summary

Increased levels of CAE parallel processing by utilizing GPUs in an HPC environment is enabling much larger and complex CAE simulations to be addressed in product development workflows. As CAE simulation requirements continue to grow such as the need for transients, high-resolutions, and multidiscipline simulation that are heavy in numerical operations, multi-level parallel systems and will be essential technologies. The heterogeneous nature of such high fidelity simulations and their HPC resource usage will continue to grow the requirements for balanced GPU-based servers within distributed memory clusters. It was demonstrated that substantial LS-DYNA performance gains can be achieved by using the latest GPU technology, the NVIDIA Tesla 20-series. Based on these trends, we can expect that HPC will be a powerful tool in the future of scientific computing and advanced CAE modeling and practice.

6. Literature

[1] Andrew C., Fernando C., Rainald L., John W., 19th AIAA Computational Fluid Dynamics, June 22-25,San Antonio, Texas.

[2] Lucas, R., Wagenbreth, G., Davis, D.: Implementing a GPU-Enhanced Cluster for Large Scale Simulations, I/ITSEC, 2007, Orlando, FL, USA.

[3] LS-DYNA User's Manual Version 971, Livermore Software Technology Corporation, Livermore, CA, 2007.

[4] Grimes, R., Lucas, R., Wagenbreth, G.: The Potential Impact of GPUs on LS-DYNA Implicit, 11th International LS-DYNA Users Conference, June 2010, Dearborn, MI, USA.

[5] NVIDIA Corporation, NVIDIA CUDA Compute Unified Device Architecture 2.0 Programming Guide, 2008.

[6] S. Kodiyalam, M. Kremenetsky and S. Posey, “Balanced HPC Infrastructure for CFD and associated Multidiscipline Simulations of Engineering Systems,” Proceedings, 7th Asia CFD Conference 2007, Bangalore, India, November 26 – 30, 2007.